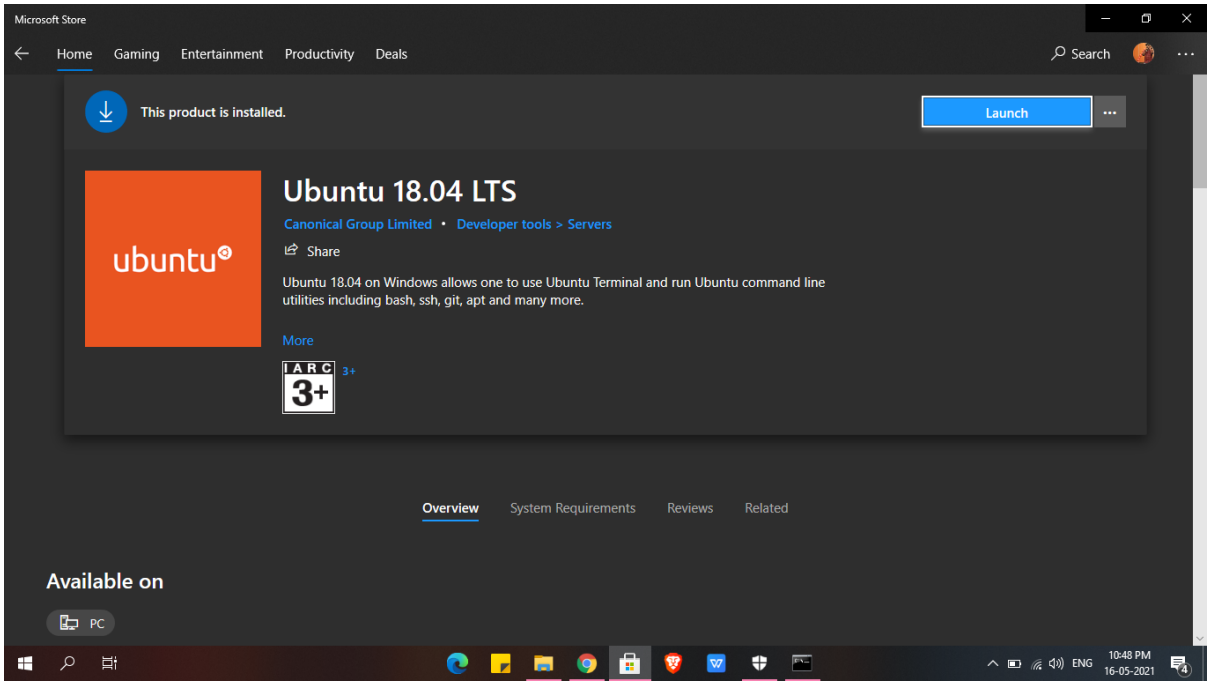
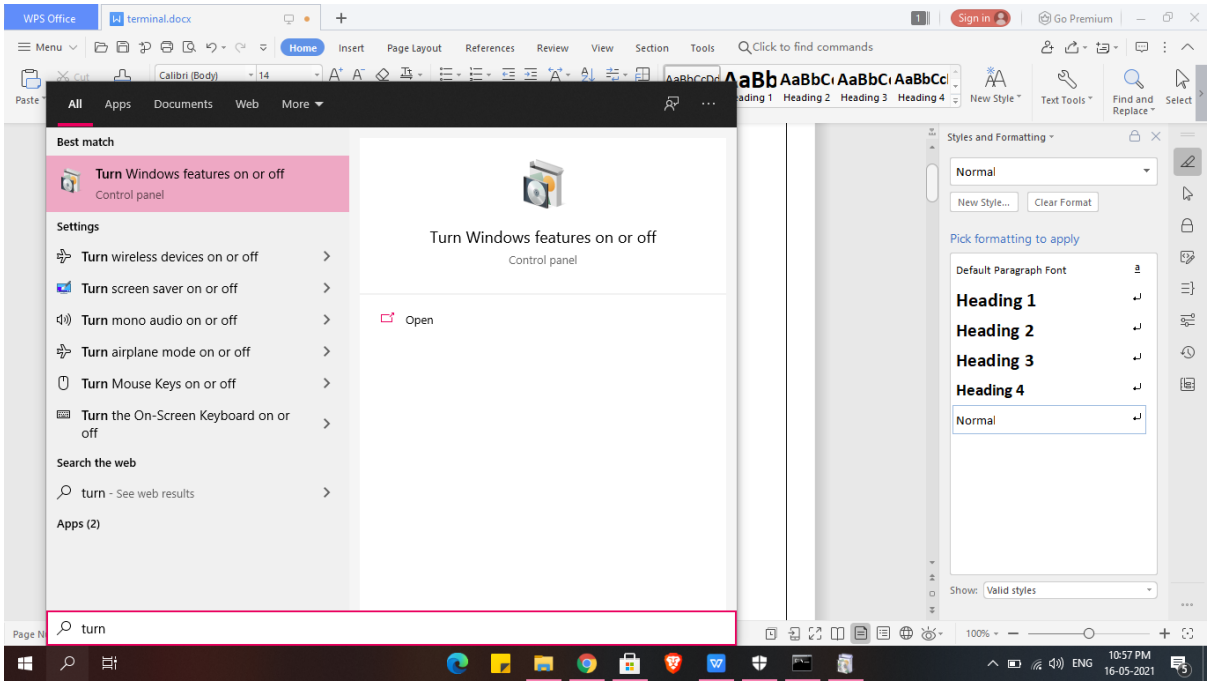
TERMINAL COMMANDS

- **Installation of wsl in windows**

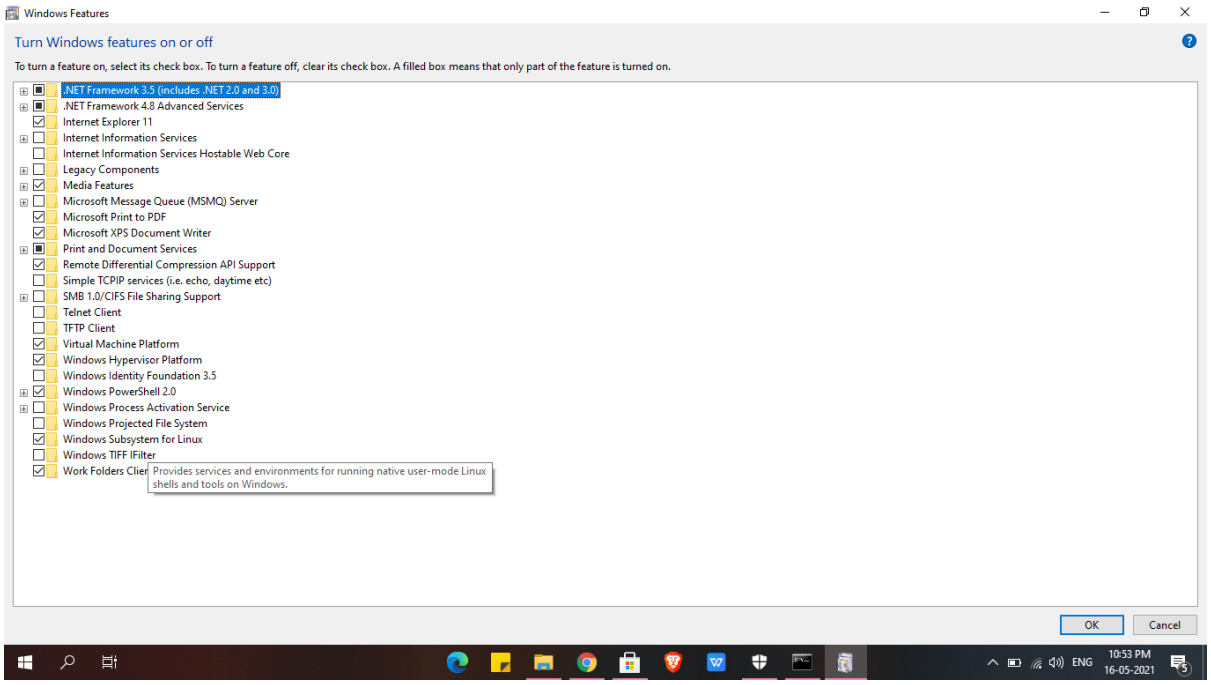
First get to the windows store and install ubuntu18.0 LTS



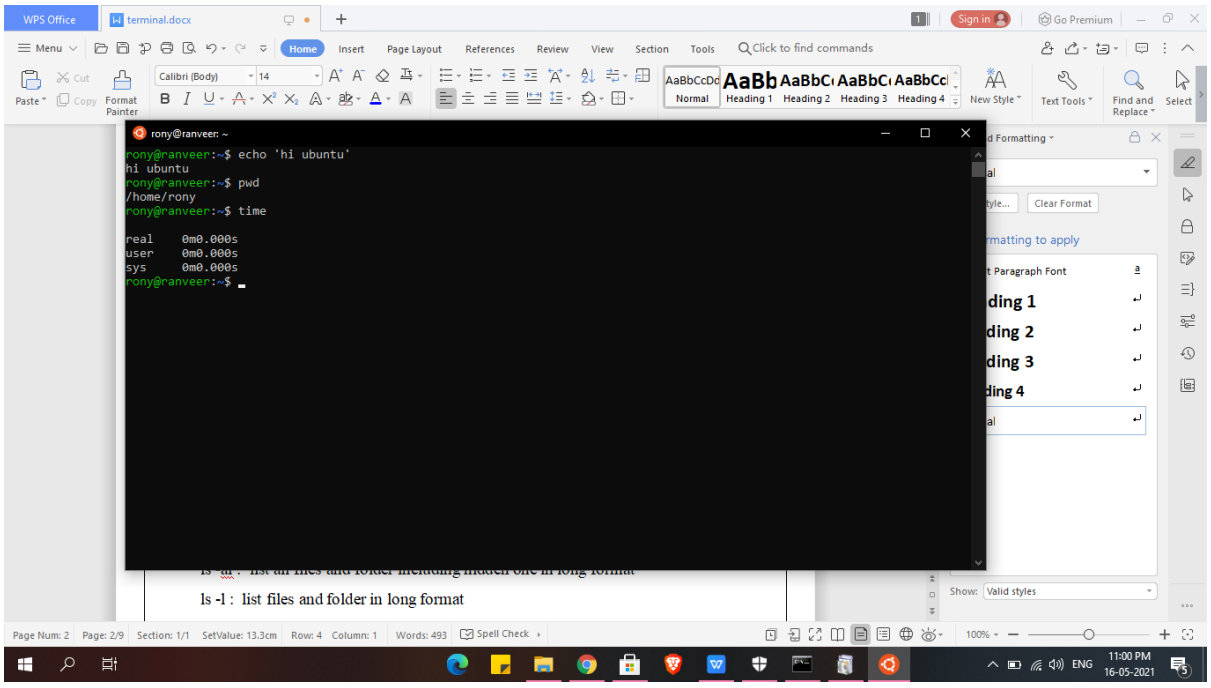
After installing it make sure u enable the wsl from turn on and off



Enabling windows subsystem for linux



Now u can search ubuntu 18.0 LTS and use it.



- ls

ls ~: gives/jump all file present in system or all home directory

ls .. : give content or file present in parent directory

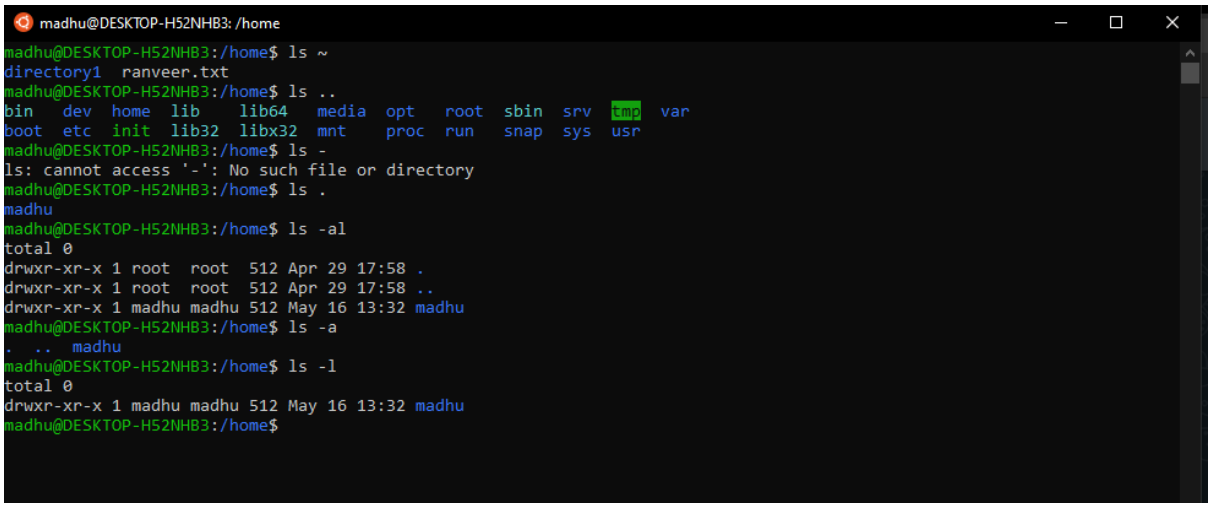
ls . : List files and folder in current directory

ls ~ : show u all content in home directory

ls -a : list all files and folder including hidden one

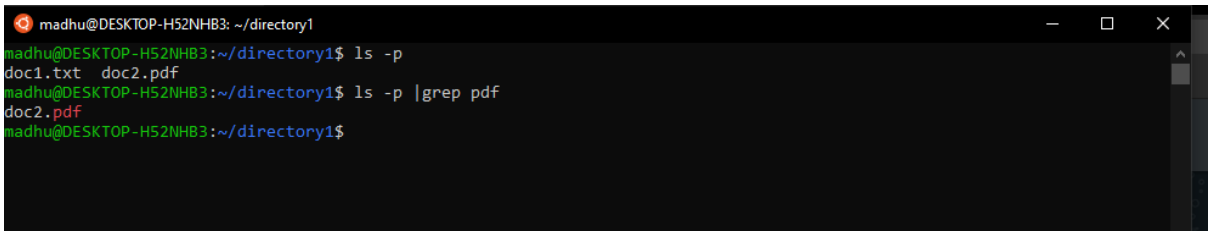
ls -al : list all files and folder including hidden one in long format

ls -l : list files and folder in long format



ls -p :give all files in current directory

ls -p | grep pdf :gives if any pdf file is present (you can write whatever type of file in place of pdf like txt n all)



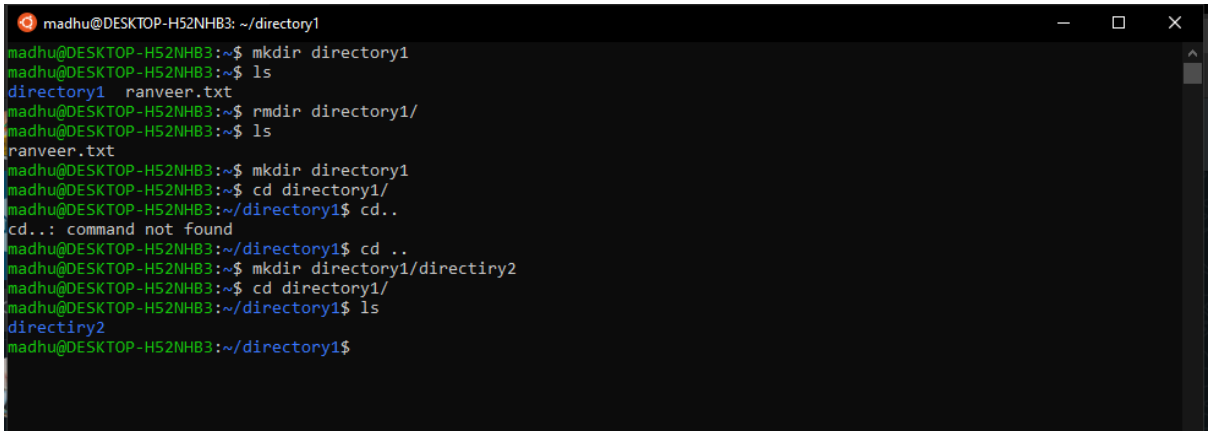
- mkdir

mkdir directory_name : create directory

rmdir dire_name : removes that particular directory

mkdir -p directory_name1/directory_name2 : create nested directory

(2 is inside 1)



- cd

cd directory_name/ : take u to that directory

cd .. : take u to one directory before

cd - : take u to previous working directory

- touch

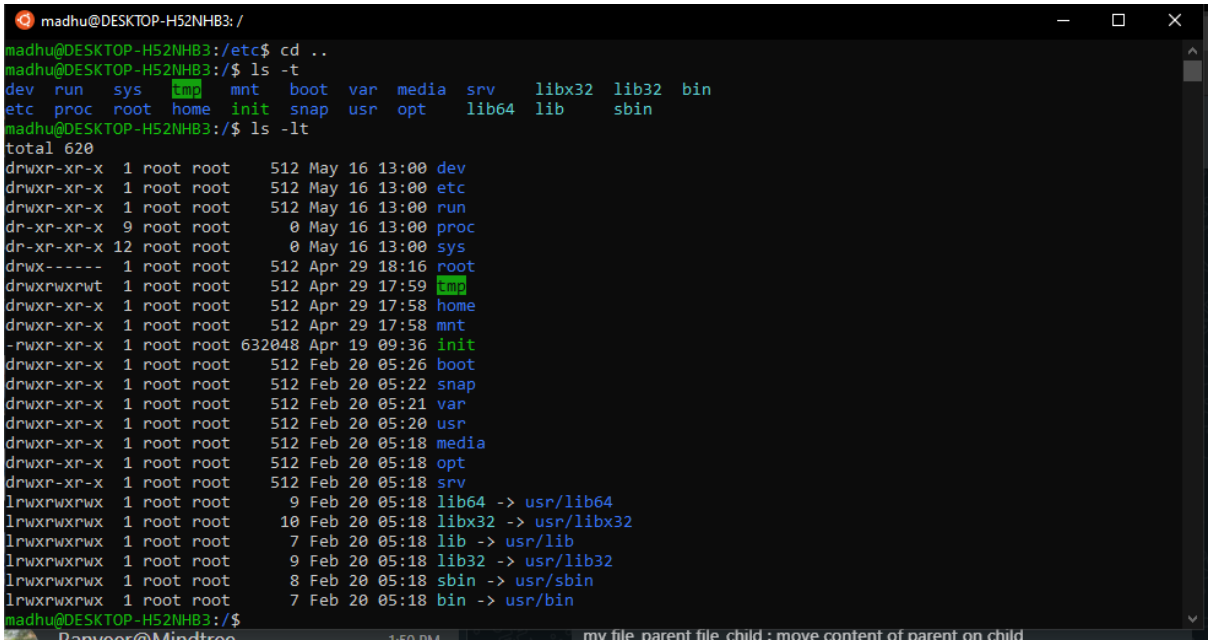
touch file_name : create empty file

cd /etc/ :take u to home directory directly

ls -t : gives highlight on last modified directory

ls -lt :give complete details about recent modified file plus other file present

touch file_name : create file name of file_name



- **rm**

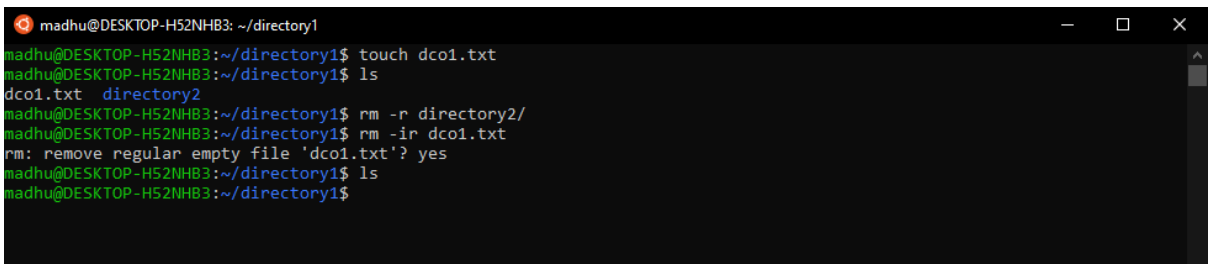
rm -r file_name : deletes that perticular file_name

rm -i file_name : ask user permission before deletion

rm -I file_name : dosnt ask user before deletion

rm -r directory_name: remove the directory recursively

rm -ir directory_name : removes the file or dir with asking permission



ls -s file_name : gives all file presnt inside in tabular format

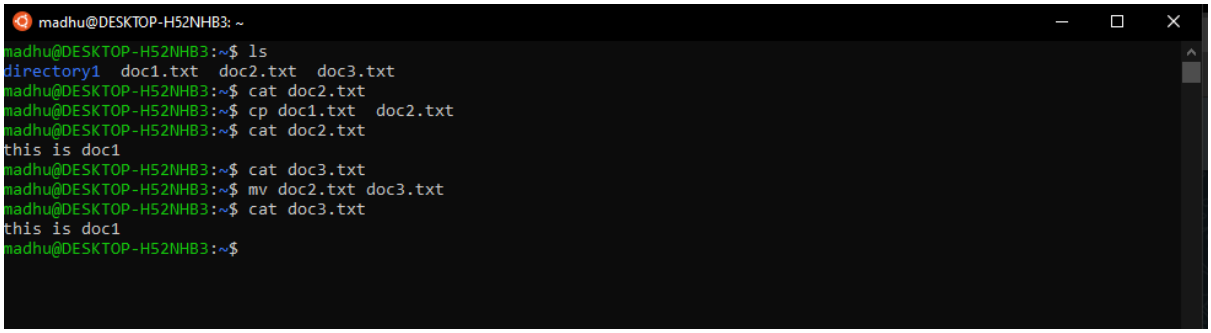
ls -l file_name : gives all file presnt inside in long vertical format

pwd : present working directory

- **CP & MV**

cp file_parent file_child : copy content of parent on child

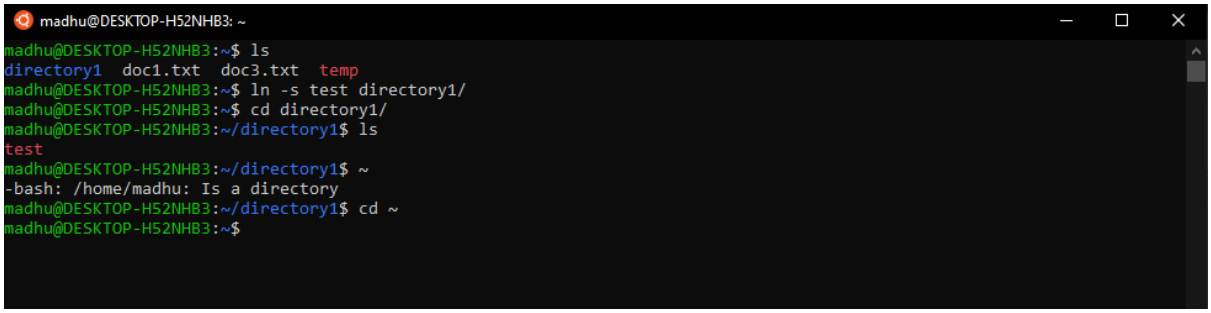
mv file_parent file_child : move content of parent on child



- **ln**

ln -s test temp/ : symbolic link test file into temp dir // ln command is used to link files with dir

~ : wherever this symbol is used it wil always take u to home always



if u want to check which file is linked in that particular dir:

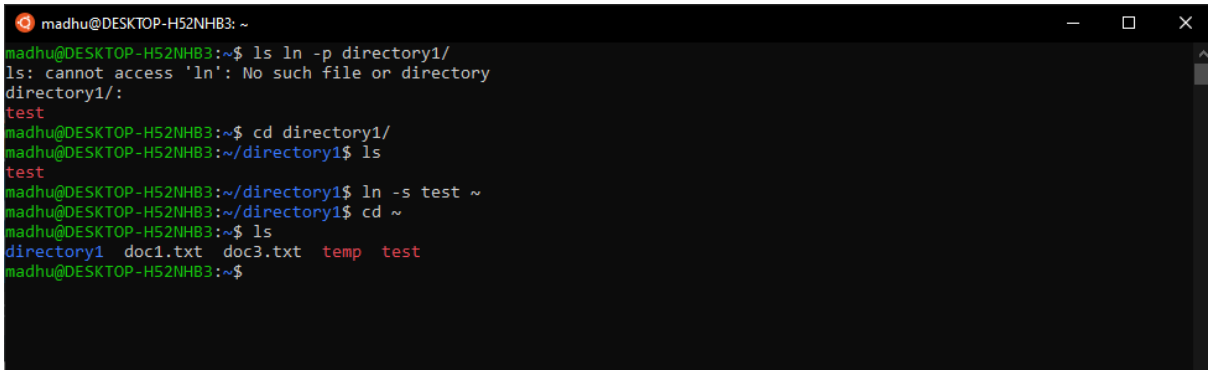
ls ln -p temp1 : temp1 is dir name

output:

temp1:

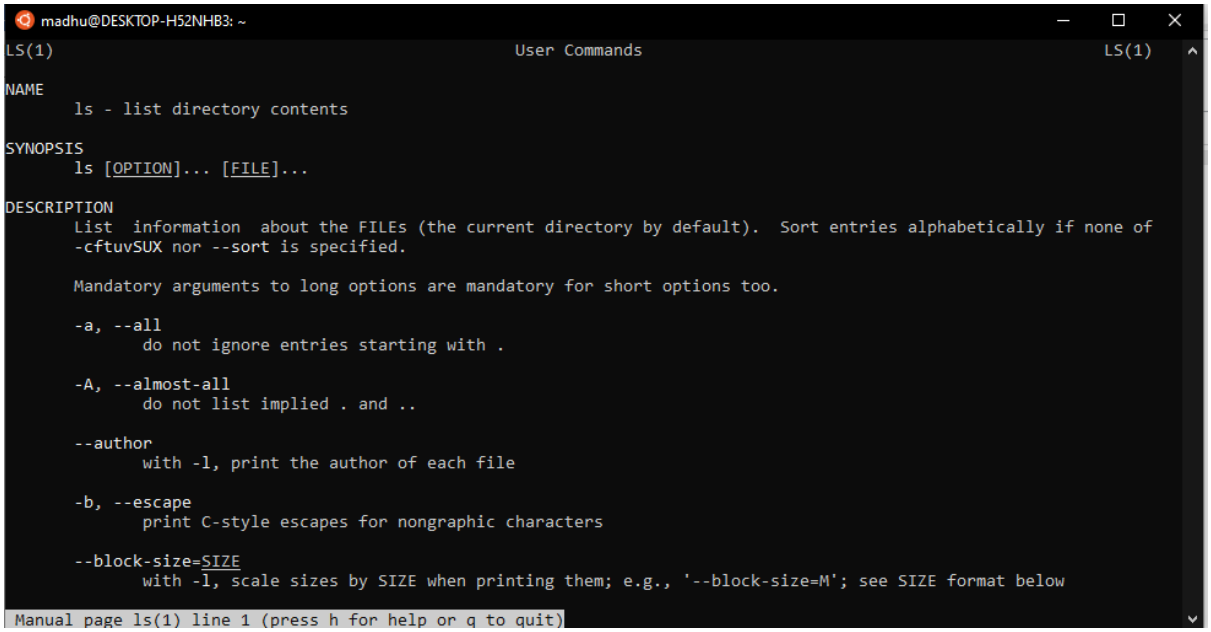
test

ln -s file_name ~/home_dir/ : link file_name to home_dir



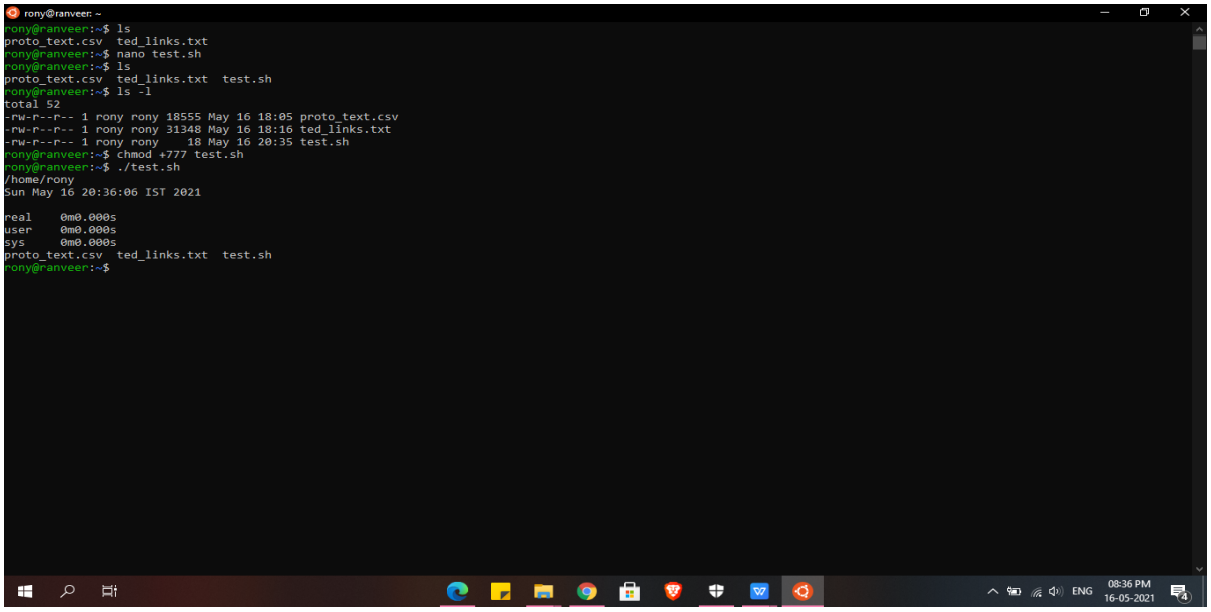
- **Man**

man command_name : gives complete details about that perticular command_name



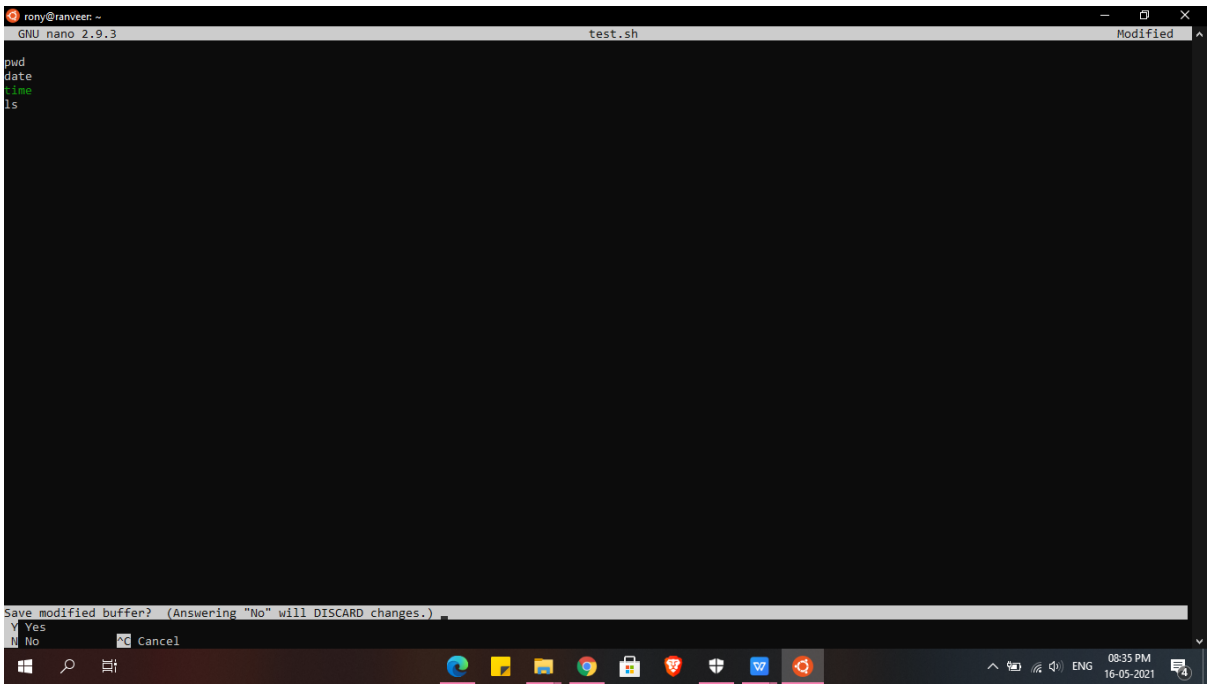
- **nano**

for making shell script we use nano command in terminal



Inside nano test.sh

Use ctrl+x to exit with y to save .



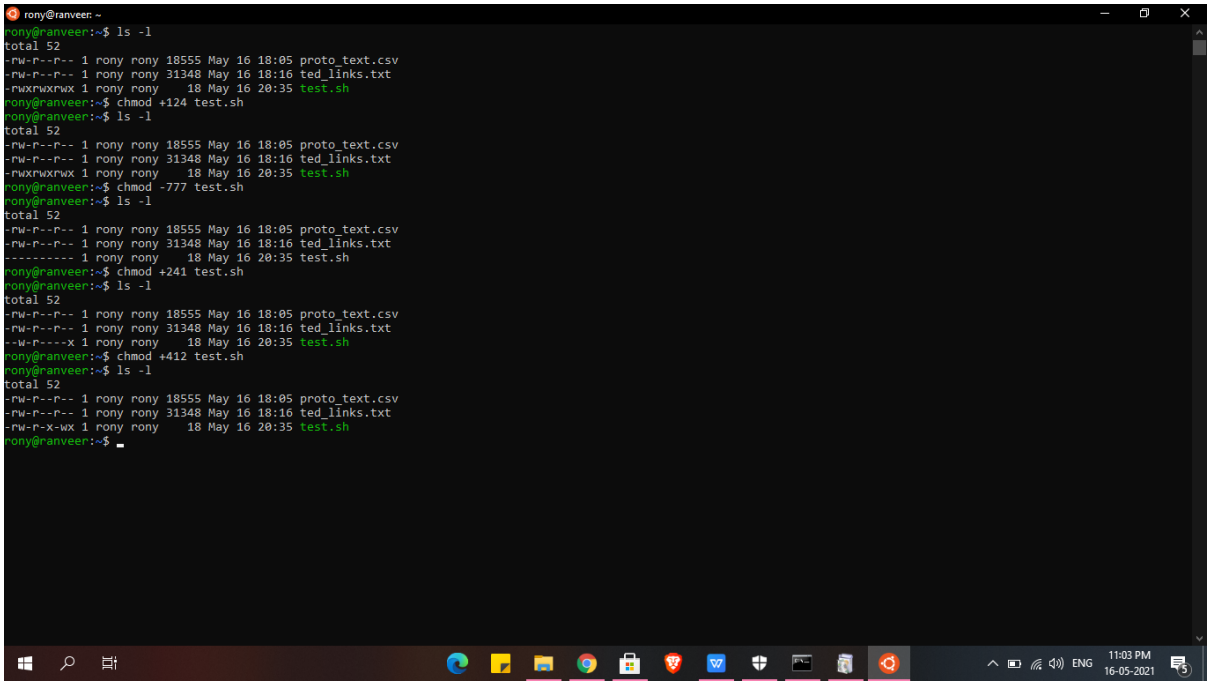
- **Chmod**

This command is used for changing the permission ..what read write and execution feature can be given to user group and owner

+1 for read

+2 for write

+4 for execute



Capability 1: Ability to obtain data using command line from various resources like CSV, JSON, XML etc..

1.1 Obtaining data from internet

- The Internet provides without a doubt the largest resource for data. This data is available in various forms, using various protocols. The command-line tool cURL is used for downloading data from the Internet.

- The easiest invocation of cURL is to simply specify a URL as a command-line argument.

```
$ curl -s http://www.gutenberg.org/files/76/76-0.txt | head -n 10

The Project Gutenberg EBook of Adventures of Huckleberry Finn, Complete
by Mark Twain (Samuel Clemens)

This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever. You may copy it, give it away or re-use
it under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.net
```

- By default, cURL outputs a progress meter that shows how the download rate and the expected time of completion.

```
$ curl http://www.gutenberg.org/files/76/76-0.txt | head -n 10
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--

The Project Gutenberg EBook of Adventures of Huckleberry Finn, Complete
by Mark Twain (Samuel Clemens)

This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever. You may copy it, give it away or re-use
it under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.net
```

- If you save the data to a file, then you do not need to necessarily specify the `-s` option:

```
$ curl http://www.gutenberg.org/files/76/76-0.txt > data/finn.txt
```

- To download from an FTP server, we use cURL in exactly the same way. When the URL is password protected, you can specify a username and a password as follows:

```
$ curl -u username:password ftp://host/file
```

- `-L` or `--location` option in order to be redirected:

```
$ curl -L j.mp/locatbbar
```

- So, cURL is a straight-forward command-line tool for downloading data from the Internet. Its three most common command-line arguments are `-s` to suppress the progress meter, `-u` to specify a username and password, and `-L` to automatically follow redirects.

1.2 Querying Database

- Most companies store their data in a relational database. Examples of relational databases are MySQL, PostgreSQL, and SQLite. These databases all have a slightly different way of interfacing with them. Some provide a command-line tool or a command-line interface, while others do not.

- A command-line tool called `sql2csv`, which is part of the Csvkit suite gives the output in CSV format.

- We can obtain data from relational databases by executing a `SELECT` query on them. (`sql2csv` also support `INSERT`, `UPDATE`, and `DELETE` queries) To select a specific set of data from an SQLite database named *iris.db*, `sql2csv` can be invoked as follows:

```
$ sql2csv --db 'sqlite:///data/iris.db' --query 'SELECT * FROM iris '\
> 'WHERE sepal_length > 7.5'
sepal_length,sepal_width,petal_length,petal_width,species
7.6,3.0,6.6,2.1,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
```

  The `--db` option sp ecifies the database URL

1.3 Connecting to Web APIs

- Data can come from the Internet is through a web API, which stands for *Application Programming Interface*.

- Web APIs are not meant to be presented in nice layout, such as websites. Instead, most web APIs return data in a structured format, such as JSON or XML. Having data in a structured form has the advantage that the data can be easily processed by other tools, such as jq. For example, the API from https://randomuser.me returns data in the following JSON structure.

```
$ curl -s https://randomuser.me/api/1.2/ | jq .
{
  "results": [
    {
      "gender": "male",
      "name": {
        "title": "mr",
        "first": "jeffrey",
        "last": "lawson"
      },
      "location": {
        "street": "838 miller ave",
        "city": "washington",
        "state": "maryland",
        "postcode": 81831,
        "coordinates": {
          "latitude": "81.9488",
          "longitude": "-67.8247"
        },
        "timezone": {
          "offset": "+4:00",
          "description": "Abu Dhabi, Muscat, Baku, Tbilisi"
        }
      },
...
```

  The data is piped to a command-line tool jq in order to display it in a nice way.

- Some web APIs return data in a streaming manner. This means that once you connect to it, the data will continue to pour in forever. A well-known example is the Twitter "firehose", which constantly streams all the tweets being sent around the world.

- Command-line tool called curlicue assists APIs require you to log in using the OAuth protocol. Once this has been set up, curlicue will call curl with the correct headers. First, we set things up once for a particular API with curlicue-setup, and then you can call that API using curlicue. For example, to use curlicue with the Twitter API you would run:

```
$ curlicue-setup \
> 'https://api.twitter.com/oauth/request_token' \
> 'https://api.twitter.com/oauth/authorize?oauth_token=$oauth_token' \
> 'https://api.twitter.com/oauth/access_token' \
> credentials
$ curlicue -f credentials \
> 'https://api.twitter.com/1/statuses/home_timeline.xml'
```

Capability 2:Ability to create reusable command line tools

- we use a lot of commands and pipelines that basically fit on one line.
- Let us call those one-liners. Being able to perform complex tasks with just a one-liner is what makes the command line powerful.
- It's a very different experience from writing traditional programs.
- If you foresee or notice that you need to repeat a certain one-liner on a regular basis, it is worthwhile to turn this into a command-line tool of its own.
- The advantage of a command-line tool is that you do not have to remember the entire one-liner and that it improves readability if you include it into some other pipeline.
- The benefit of a working with a programming language, however, is that you have the code in a file. This means that you can easily reuse that code
- In order to turn a one-liner into a shell script, we need to use some shell scripting.

KO1: Converting One-liners into Shell Scripts

```
$ curl -s http://www.gutenberg.org/files/76/76-0.txt |
> tr '[:upper:]' '[:lower:]' |
> grep -oE '\w+' |
> sort |
> uniq -c |
> sort -nr |
> head -n 10
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
   2567 it
   2086 t
   2044 was
   1847 he
   1778 of
```

This one-liner returns the top ten words of the e-book version of *Adventures of Huckleberry Finn*

Explaining the action of Each One-Liner.

- Downloading an ebook using ***curl***
- Converting the entire text to lowercase using ***tr***
- Extracting all the words using ***grep***
- Sort these words in alphabetical order using ***sort***
- Remove all the duplicates and count how often each word appears in the list using ***uniq***
- Sort this list of unique words by their count in descending order using ***sort***
- Keep only the top 10 lines (i.e., words) using ***head***

Step 1: Write the code into a shell file

a) The first step is to create a new file.
b) Open your favorite text editor and copy and paste our one-liner.
c) Name the file *top-words-1.sh*

d) Use bash to interpret and execute the commands in the file

```
curl -s http://www.gutenberg.org/files/76/76-0.txt |
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

In order to compare differences between steps, we copy the file to *top-words-2.sh* using `cp top-words-{1,2}.sh` . You can keep working with the same file if you want to.

Step 2: Add Permission to Execute

a) To change the access permissions of a file, we need to use a command-line tool called **chmod**
b) **chmod** stands for *change mode*

```
$ chmod u+x top-words-2.sh
```

- The command-line argument **u+x** consist of 3 arguments
  - ✓ **u** indicates that we want to change the permissions for the user who owns the file, which is you, because you created the file
  - ✓ **+** indicates that we want to add a permission
  - ✓ **x** indicates the permissions to execute

```
$ ls -l top-words-{1,2}.sh
-rw-rw-r-- 1 vagrant vagrant 145 Jul 20 23:33 top-words-1.sh
-rwxrw-r-- 1 vagrant vagrant 143 Jul 20 23:34 top-words-2.sh
```

Now we can see the difference of permissions b/w the 2 files.

- Now you can execute the file as follows

```
$ ./top-words-2.sh
```

```
6441 and
5082 the
3666 i
3258 a
3022 to
2567 it
2086 t
2044 was
1847 he
1778 of
```

Step 3: Define Shebang

a) we should add a so-called *shebang* to the file.
b) The shebang is a special line in the script, which instructs the system which executable should be used to interpret the commands.
c) We want to use **bash** to interpret our commands.
d) We can copy contents from top-word2.sh to top-word3.sh to see the difference.

```
#!/usr/bin/env bash
curl -s http://www.gutenberg.org/files/76/76-0.txt |
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

Step 4: Remove Fixed Input

a) We can make our command-line tool more reusable.
b) The user of the command-line tool will provide the text, it will become generally applicable
c) Remove the **curl** command from file.

```
#!/usr/bin/env bash
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

- This works because if a script starts with a command that needs data from standard input, **tr** it will take the input that is given to the command-line tools
- Also copy paste contents from file in set 3 to top-word-4.sh

```
$ cat data/finn.txt | top-words-4.sh
```

- Here we have the input text file in finn.txt that is given as the data input to our cmd tool shell script file top-word-4.sh

Step 5: Parametrize

a) There is one more step that we can perform in order to make our command-line tool even more reusable: parameters.
b) It would be very useful to allow for different values for the **head** commands.
c) This would allow the end user to set the number of most-often used words to be outputted.
d) Write the below code to new file top-word-5.sh

```
#!/usr/bin/env bash
NUM_WORDS="$1"
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n $NUM_WORDS
```

- The variable *NUM_WORDS* is set to the value of *$1*, which is a special variable in Bash. It holds the value of the first command-line argument passed to our command-line tool.
- Using the below command you user can provide the number of words as a parameter while executing the script.

```
$ cat data/finn.txt | top-words-5 5
```

KO2: Creating Command-line Tools with Python and R

Porting the prior shell scripts to Python and R

Lets see how the same task to be done in Python and R

```
#!/usr/bin/env bash
NUM_WORDS="$1"
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n $NUM_WORDS
```

Code in Shell Script

```
#!/usr/bin/env python
import re
import sys
from collections import Counter
num_words = int(sys.argv[1])
text = sys.stdin.read().lower()
words = re.split('\W+', text)
cnt = Counter(words)
for word, count in cnt.most_common(num_words):
    print "%7d %s" % (count, word)
```

Code in Python

```
#!/usr/bin/env Rscript
n <- as.integer(commandArgs(trailingOnly = TRUE))
f <- file("stdin")
lines <- readLines(f)
words <- tolower(unlist(strsplit(lines, "\\W+")))
counts <- sort(table(words), decreasing = TRUE)
counts_n <- counts[1:n]
cat(sprintf("%7d %s\n", counts_n, names(counts_n)), sep = "")
close(f)
```

Code in R

Outputs after executing the files.

```
$ < data/76.txt top-words.sh 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
$ < data/76.txt top-words.py 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
$ < data/76.txt top-words.R 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
```

Processing Streaming Data from Standard Input

- Luckily Python and R support processing streaming data.
- You can apply a function on a line-per-line basis
- So you doesn't need to provide the input completely at the beginning.

```python
#!/usr/bin/env python
from sys import stdin, stdout
while True:
    line = stdin.readline()
    if not line:
        break
    stdout.write("%d\n" % int(line)**2)
    stdout.flush()
```

Code in Python

```r
#!/usr/bin/env Rscript
f <- file("stdin")
open(f)
while(length(line <- readLines(f, n = 1)) > 0) {
        write(as.integer(line)^2, stdout())
}
close(f)
```

Code in R

Capability 3: Ability to apply scrubbing operations using command line operators.

It is not uncommon to have missing values, inconsistencies, errors, weird characters, etc. when obtaining data from a variety of sources. In those cases, we must scrub, or clean, the data before we explore them.

- Once our data is in the format, we want it to be, we can apply common scrubbing operations. These include filtering, replacing, and merging data. The command line is especially well-suited for these kinds of operations, as there exist many powerful command-line tools.

- If your data requires additional functionality than that is offered by (a combination of) these command-line tools, you can use `csvsql`. This is a new command-line tool that allow you to perform SQL queries directly on CSV files.

3.1 Scrubbing operations

3.1.1 Filtering lines

- The first scrubbing operation is filtering lines where each line from the input data will be evaluated whether it may be passed on as output.

- **Based on location**: The most straightforward way to filter lines is based on their location. This may be useful when you want to inspect, say, the top 10 lines of a file, or when you extract a specific row from the output of another command-line tool.

Eg1: The top 10 lines of a file:

```
$ seq -f "Line %g" 10 | tee data/lines
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

Eg2: print the first 3 lines using either head, sed, or awk:

```
$ < lines head -n 3
$ < lines sed -n '1,3p'
```

```
$ < lines awk 'NR<=3'
Line 1
Line 2
Line 3
```

Eg3: print the last 3 lines using **tail:**

```
$ < lines tail -n 3
Line 8
Line 9
Line 10
```

- **Based on pattern** : Using `grep`, the canonical command-line tool for filtering lines, we can print every line that matches a certain pattern or regular expression.

  - to extract all the chapter headings from *Alice's Adventures in Wonderland*:

```
$ grep -i chapter alice.txt
CHAPTER I. Down the Rabbit-Hole
CHAPTER II. The Pool of Tears
CHAPTER III. A Caucus-Race and a Long Tale
CHAPTER IV. The Rabbit Sends in a Little Bill
CHAPTER V. Advice from a Caterpillar
CHAPTER VI. Pig and Pepper
```

- **Based on randomness** : The command-line tool `sample` is to get a subset of the data by outputting only a certain percentage of the input on a line-by-line basis.

```
$ seq 1000 | sample -r 1% | jq -c '{line: .}'
{"line":53}
{"line":119}
{"line":141}
{"line":228}
{"line":464}
{"line":476}
{"line":523}
{"line":657}
{"line":675}
{"line":865}
{"line":948}
```

3.1.2 Extracting Lines

- To extract the actual chapter headings from our example earlier, we can take a simple approach by piping the output of `grep` to `cut`:

```
$ grep -i chapter alice.txt | cut -d' ' -f3-
```

```
Down the Rabbit-Hole
The Pool of Tears
A Caucus-Race and a Long Tale
The Rabbit Sends in a Little Bill
Advice from a Caterpillar
Pig and Pepper
A Mad Tea-Party
The Queen's Croquet-Ground
The Mock Turtle's Story
The Lobster Quadrille
Who Stole the Tarts?
Alice's Evidence
```

3.1.3 Replacing and deleting values

- the command-line tool `tr`, which stands for translate, to replace individual characters. For example, spaces can be replaced by underscores as follows:

```
$ echo 'hello world!' | tr ' ' '_'
hello_world!
```

- If more than one character needs to be replaced, then you can combine that:

```
$ echo 'hello world!' | tr ' !' '_?'
hello_world?
```

- to change a word, remove repeated spaces, and remove leading spaces:

```
$ echo ' hello     world!' | sed -re 's/hello/bye/;s/\s+/
/g;s/\s+//'
bye world!
```

The argument `-g` stands for global, meaning that the same command can be applied more than once on the same line.

3.2 Convert data from one format to another

- The data can come in a variety of formats, the most common ones are plain text, CSV, JSON, and HTML/XML. Since most command-line tools operate on one format only, it is worthwhile to be able to convert data from one format to another.
- There are two reasons to convert data:
    1. First, for visualization and machine learning algorithms, the data needs to be in tabular form, just like a database table or a spreadsheet. CSV is inherently in

tabular form, but JSON and HTML/XML data can have a deeply nested structure.

2. Second, many command-line tools, especially the classic ones such as `cut` and `grep`, operate on plain text. This is because text is regarded as a universal interface between command-line tools. Other formats can be treated as plain text, allowing us to apply such command-line tools to the other formats as well.

3. Example: changing the attribute *gender* to *sex* using `sed`:

    - 'sed' command stands for stream editor. It is used to edit streams (files) using regular expressions. This editing remains only in display, whereas file content remains the same.

    - ```
      $ sed -e 's/"gender":/"sex":/g' data/users.json | fold | head -n 3
      ```

    - `sed` does not make use of the structure of the data. So first convert the data to a tabular format such as CSV and then apply the appropriate command-line tool.

- To convert XML/HTML and JSON to CSV through a real-world use case, the command-line tools required are: `curl`, `scrape` , `xml2json` , `jq`, and `json2csv`.

    1. The very first step is to download the HTML using `curl`. The option `-s` causes `curl` to be silent and not output any other information but the actual HTML. The HTML is saved to a file named `data/wiki.html`

    ```
    $ curl -sL 'http://en.wikipedia.org/wiki/List_of_countries_and_territories_'\
    > 'by_border/area_ratio' > data/wiki.html
    ```

    2. The next step is to extract the necessary elements from the HTML file. For this we use the `scrape` tool. The value passed to argument `-e`, which stands for *expression*. The syntax is usually used to style web pages, but we can also use it to select certain elements from our HTML.

    ```
    $ < data/wiki.html scrape -b -e 'table.wikitable > tr:not(:first-child)' \
    > > data/table.html
    $ head -n 21 data/table.html
    <!DOCTYPE html>
    <html>
    <body>
    <tr><td>1</td>
    <td>Vatican City</td>
    <td>3.2</td>
    ```
    ```

```
    <td>0.44</td>
    <td>7.2727273</td>
    </tr>
```

3. As the name implies, `xml2json` converts XML (and HTML) to JSON

```
$ < data/table.html xml2json > data/table.json
$ < data/table.json jq '.' | head -n 25
{
"html": {
"body": {
"tr": [
{
"td": [
{
"$t": "1"
},
{
"$t": "Vatican City"
},
{
"$t": "3.2"
},
{
"$t": "0.44"
},
{
"$t": "7.2727273"
}
]
},
{
"td": [
```

4. The tool `json2csv` is used to convert the data from JSON to CSV:

```
$ < data/countries.json json2csv -p -k border,surface >
data/countries.csv
$ head -n 11 data/countries.csv | csvlook
|---------+----------|
|  border | surface  |
|---------+----------|
|  3.2    | 0.44     |
|  4.4    | 2        |
|  39     | 61       |
|  76     | 160      |
|  10.2   | 34       |
|  120.3  | 468      |
|  1.2    | 6        |
|  10.2   | 54       |
|  359    | 2586     |
|  466    | 6220     |
|---------+----------|
```

3.3 Working with CSV

- In order to leverage ordinary command-line tools for CSV, we'd like to introduce you to three command-line tools, aptly named: `body`, `header`, and `cols`.
- With `body` we can apply any command-line tool to the body of a CSV file, that is, everything excluding the header.

```
$ echo -e "value\n7\n2\n5\n3" | body sort -n
value
2
3
5
7
```

- If no argument are provided, the header of the CSV file is printed:

```
$ < tips.csv | header
bill,tip,sex,smoker,day,time,size
```

- we count the lines of the following CSV file:

```
$ seq 5 | header -a count
count
1
2
3
4
5
```

- With `wc -l`, we can count the number of all lines:

```
$ seq 5 | header -a count | wc -l
6
```

Performing SQL queries on csv

- The command-line tool `csvsql` allows you to execute SQL queries directly on CSV files.
- The basic command is :

```
$ seq 5 | header -a value | csvsql --query "SELECT SUM(value) AS sum FROM
stdin"
sum
15
```

- SQL approach for extracting and reordering the numerical columns of an Iris data set:

```
$ < iris.csv csvsql --query "SELECT sepal_length, petal_length, "\
> "sepal_width, petal_width FROM stdin" | head -n 5 | csvlook
```

```
|---------------+--------------+-------------+--------------|
|  sepal_length | petal_length | sepal_width | petal_width  |
|---------------+--------------+-------------+--------------|
|  5.1          | 1.4          | 3.5         | 0.2          |
|  4.9          | 1.4          | 3.0         | 0.2          |
|  4.7          | 1.3          | 3.2         | 0.2          |
|  4.6          | 1.5          | 3.1         | 0.2          |
|---------------+--------------+-------------+--------------|
```

- The csvsql solution is more verbose but is also more robust as it uses the names of the columns instead of their indexes:

```
$ < tips.csv csvsql --query "SELECT * FROM stdin "\
> "WHERE bill > 40 AND day LIKE '%S%'" | csvlook -I
|--------+------+--------+--------+-----+--------+-------|
|  bill  | tip  | sex    | smoker | day | time   | size  |
|--------+------+--------+--------+-----+--------+-------|
|  48.27 | 6.73 | Male   | 0      | Sat | Dinner | 4     |
|  44.3  | 2.5  | Female | 1      | Sat | Dinner | 3     |
|  48.17 | 5.0  | Male   | 0      | Sun | Dinner | 6     |
|  50.81 | 10.0 | Male   | 1      | Sat | Dinner | 3     |
|  45.35 | 3.5  | Male   | 1      | Sun | Dinner | 3     |
|  40.55 | 3.0  | Male   | 1      | Sun | Dinner | 2     |
|  48.33 | 9.0  | Male   | 0      | Sat | Dinner | 4     |
|--------+------+--------+--------+-----+--------+-------|
```

Capability 4: Managing Your Data Workflow

- The command line is a very convenient environment for exploratory data analysis
- As this process is of an exploratory nature, our workflow tends to be rather chaotic, which makes it difficult to keep track of what we've done.
- It is very important that our steps can be reproduced, whether that is by ourselves or by others
- A better approach would be to save your commands to a shell script *run.sh*.
- A shell script is, however, a sub-optimal approach.

- This is where Drake comes in handy (Factual 2014).
- Drake is command-line tool created by Factual that allows you to:
  - Formalize your data workflow steps in terms of input and output dependencies.
  - Run specific steps of your workflow from the command line.
  - Use inline code.
  - Store and retrieve data from external sources

Introducing Drake

i. Drake organizes command execution around data and its dependencies.
ii. Your data processing steps are formalized in a separate text file (a workflow).
iii. Each step usually has one or more inputs and outputs.
iv. Drake automatically resolves their dependencies and determines which commands need to be run and in which order.

Installing Drake

Drake is written in the programming language Clojure which means that it runs on the Java Virtual Machine (JVM). There are pre-built jars available but because Drake is in active development, we will build it from source. For this, you will need to install Leiningen.

```
$ sudo apt-get install openjdk-6-jdk
$ sudo apt-get install leiningen
```

Then, clone the Drake repository from Factual:

```
$ git clone https://github.com/Factual/drake.git
```

And build the uberjar using Leiningen:

```
$ cd drake
$ lein uberjar
```

This creates *drake.jar*. Copy this file to a directory which is on your *$PATH*, for example, *~/bin*:

```
$ mv drake.jar ~/bin/
```

At this point you should already be able to run Drake:

```
$ cd ~/bin/
$ java -jar drake.jar
```

An alternative method

```
$ git clone https://github.com/flatland/drip.git
$ cd drip
$ make prefix=~/bin install
```

Then, create a Bash script that allows you to run Drake from everywhere:

```
$ cd ~/bin
$ cat << 'EOF' > drake
> #!/bin/bash
> drip -cp $(dirname $0)/drake.jar drake.core "$@"
> EOF
$ chmod +x drake
```

To verify that you have correctly installed both Drake and Drip, you can run the following command, preferably from a different directory:

```
$ drake --version
Drake Version 0.1.6
```

Task is to Obtain Top E-Books from Gutenberg

```
$ curl -s 'http://www.gutenberg.org/browse/scores/top' |
> grep -E '^<li>' |
> head -n 5 |
> sed -E "s/.*ebooks\/([0-9]+).*/\\1/" > data/top-5
```

This command:
- Downloads the HTML.
- Extracts the list items.
- Keeps only the top five items.
- Saves e-book IDs to *data/top-5*.

Output is

```
$ cat data/top-5
1342
76
11
1661
1952
```

- Next step is convert this to workflow.
- A workflow is just a text file
- First Step: Download the html.
- Second Step: Process the html.
- Second step depends on the first step.
- We can define this dependency in our workflow

```
NUM:=5
BASE=data/

top.html <- [-timecheck]
    curl -s 'http://www.gutenberg.org/browse/scores/top' > $OUTPUT

top-$[NUM] <- top.html
    < $INPUT grep -E '^<li>' |
    head -n $[NUM] |
    sed -E "s/.*ebooks\/([0-9]+)\">([^<]+)<.*/\\1,\\2/" > $OUTPUT
```

- You can specify variables in Drake, preferably at the beginning of the file.
- First step is having the input data, that is saved into top.html.
- This output is defined again as the input of step two. This is how Drake knows that the second step depends on the first step.
- We have used two more special variables: *INPUT* and *OUTPUT*. Values of these two special variables are set to what we have defined as the input and output of that step, respectively.
- It allows us to easily reuse certain steps in any future workflows.

Let’s execute this new workflow using Drake: Copy the code to a new drake file named 02.drake

```
$ drake -w 02.drake
The following steps will be run, in order:
  1: ../../data/top.html <-  [missing output]
  2: ../../data/top-5 <- ../../data/top.html [projected timestamped]
Confirm? [y/n] y
Running 2 steps with concurrence of 1...

--- 0. Running (missing output): ../../data/top.html <-
--- 0: ../../data/top.html <-  -> done in 0.89s
--- 1. Running (missing output): ../../data/top-5 <- ../../data/top.html
--- 1: ../../data/top-5 <- ../../data/top.html -> done in 0.02s
Done (2 steps run).
```

Now, let’s assume that we want instead of the top 5 e-books, the top 10 e-books. We can set the *NUM* variable from the command line and run Drake again

```
$ NUM=10 drake -w 02.drake
The following steps will be run, in order:
  1: ../../data/top-10 <- ../../data/top.html [missing output]
Confirm? [y/n] y
Running 1 steps with concurrence of 1...

--- 1. Running (missing output): ../../data/top-10 <- ../../data/top.html
--- 1: ../../data/top-10 <- ../../data/top.html -> done in 0.02s
Done (1 steps run).
```

- Drake allows us to run certain steps again

```
$ drake -w 02.drake '=top.html'
```

- There is a more convenient way than using the output filename to specify which step you want to execute again

We can add so-called *tags* to both the input and output of steps. A tag starts with a “%”. It is a good idea to choose a short and descriptive tag name so that you can easily specify this at the command line.

```
NUM:=5
BASE=data/

top.html, %html <- [-timecheck]
   curl -s 'http://www.gutenberg.org/browse/scores/top' > $OUTPUT

top-$[NUM], %filter <- top.html
   < $INPUT grep -E '^<li>' |
   head -n $[NUM] |
   sed -E "s/.*ebooks\/([0-9]+)\">([^<]+)<.*/\\1,\\2/" > $OUTPUT
```

We can now rebuild the first step by specifying the *%html* tag

```
$ drake -w 03.drake '=%html'
```

Capability 5: Ability to explore data using command line operators.

- Exploring is the step where you familiarize yourself with the data.
- Being familiar with the data is essential when you want to extract any value from it.
- Exploring your data can be done from three perspectives.
    1. The first perspective is to inspect the data and its properties. Here, we want to know, for example, what the raw data looks like, how many data points the data set has, and what kind of features the data set has.
    2. The second perspective is computing descriptive statistics from data. One advantage of this perspective is that the output is often brief and textual and can therefore be printed on the command line.
    3. The third perspective is to create visualizations of the data. An advantage of visualizations over descriptive statistics is that they are more flexible and that they can convey much more information.

5.1 Inspecting Data and its properties.

5.1.1 Inspecting Headers

- To check whether your file has a header print the first few lines using:

```
$ #? [echo]
$ head file.csv | csvlook
```

- This decides whether the first line is indeed a header or already the first data point.

5.1.2 Inspect All Data

- To inspect the raw data, it's best not to use the `cat` command-line tool, since `cat` prints all the data to the screen in one go.

```
$ #? [echo]
$ less -S file.csv
```

- The `-S` command-line argument ensures that long lines are not being wrapped when they do not fit in the terminal. Instead, `less` allows you to scroll horizontally to see the rest of the lines.
- Once you are in `less`, you can scroll down a full screen by pressing `<Space>`. Scrolling horizontally is done by pressing `<Left>` and `<Right>`.
  Press `g` and `G` to go to start and the end of the file, respectively.

- If you want the data set to be nicely formatted, you can add in `csvlook`:

```
$ #? [echo]
$ < file.csv csvlook | less -S
```

5.1.3 Feature Names and Data Types

- In order to gain insight into the data set, it is useful to print the feature names and study them.

Example: -

```
$ < data/investments.csv names

company_permalink
company_name
company_category_list
company_market
…
```

- Besides the names of the columns, it would be very useful to know what type of values each column contains. For that use the following command:

Example: -

```
$ < data/datatypes.csv csvlook
|-----+--------+-------+----------+------------------+------------+-------
|  a  | b      | c     | d        | e                | f          | g
|-----+--------+-------+----------+------------------+------------+-------
|  2  | 0.0    | FALSE | "Yes!"   | 2011-11-11 11:00 | 2012-09-08 | 12:34
|  42 | 3.1415 | TRUE  | Oh, good | 2014-09-15       | 12/6/70    | 0:07PM
|  66 |        | False | 2198     |                  |            |
|-----+--------+-------+----------+------------------+------------+-------
```

5.2 Computing Descriptive Statistics

5.2.1 csvstat

- The command-line tool `csvstat` gives a lot of information:
    1. The data type in Python terminology
    2. Whether it has any missing values (nulls).
    3. The number of unique values.
    4. Various descriptive statistics (maximum, minimum, sum, mean, standard deviation, and median) for those features for which it is appropriate.

Example: -

```
$ csvstat data/datatypes.csv
  1. a
```

```
        <type 'int'>
        Nulls: False
        Values: 2, 66, 42
  2. b
        <type 'float'>
        Nulls: True
        Values: 0.0, 3.1415
  3. ....
```

- For a more concise output specify one of the statistics arguments:
    1. `--max` (maximum)
    2. `--min` (minimum)
    3. `--sum` (sum)
    4. `--mean` (mean)
    5. `--median` (median)
    6. `--stdev` (standard deviation)
    7. `--nulls` (whether column contains nulls)
    8. `--unique` (unique values)
    9. `--freq` (frequent values)
    10. `--len` (max value length)

```
Example1: -
   $ csvstat data/datatypes.csv --null
   1. a: False
   2. b: True
   3. c: False
   4. d: False
   5. e: True
   6. f: True
   7. g: True
```

- To select a subset of features with the `-c` command-line argument.

```
Example2: -
$ csvstat data/investments2.csv -c 2
2. company_name
<type 'unicode'>
Nulls: True
Unique values: 27324
5 most frequent values:
   Aviir:  13
        Galectin Therapeutics:  12
        Rostima:        12
        Facebook:       11
        Lending Club:   11
        Max length: 66
```

- To only see the relevant line, we can use `tail`:

```
        $ csvstat data/iris.csv | tail -n 1
```

Capability 6: Ability to work with parallel pipelines using command line operators

GNU Parallel allows us to apply a command or pipeline with a range of arguments such as numbers, lines, and files. Plus, it allows us to run our commands in parallel.

6.1 Able to Scrape hundreds of web pages:

- curl is a great tool to access a website's whole html code from the command line.
- By default, cURL outputs a progress meter that shows how the download rate and the expected time of completion. If you are piping the output directly to another command-line tool, such as head, be sure to specify the -s command-line argument, which stands for *silent*, so that the progress meter is disabled. Compare, for example, the output with the following command:

```
$ curl -s http://www.gutenberg.org/files/76/76-0.txt | head -n 10

The Project Gutenberg E Book of Adventures of Huckleberry Finn, Completeby Mark Twain (Samuel Clemens)

$ curl http://www.gutenberg.org/files/76/76-0.txt | head -n 10

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
Dload  Upload   Total   Spent    Left  Speed 0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--The Project Gutenberg EBook of Adventures of Huckleberry Finn, Completeby Mark Twain (Samuel Clemens)
```

```
$ curl http://www.gutenberg.org/files/76/76-0.txt > data/finn.txt
```

- You can also save the data by explicitly specifying the output file with the -o option:

```
$ curl -s http://www.gutenberg.org/files/76/76-0.txt -o data/finn.txt
```

- Steps to follow for scrapping :
  - Download the .html site with curl!
  - Extract the text with html2text!
  - Clean the data with sed, head, tail, grep or anything else you need!

EXAMPLE

1. Curl

```
curl https://www.ted.com/talks/sir_ken_robinson_do_schools_kill_creativity/transcript
```

2. Html2text

```
sudo apt-get install html2text
```

3. Cleaning

```
curl https://www.ted.com/talks/sir_ken_robinson_do_schools_kill_creativity/transcript | html2text
```

4. Removing the unnecessary parts of the data(by using the sed command)

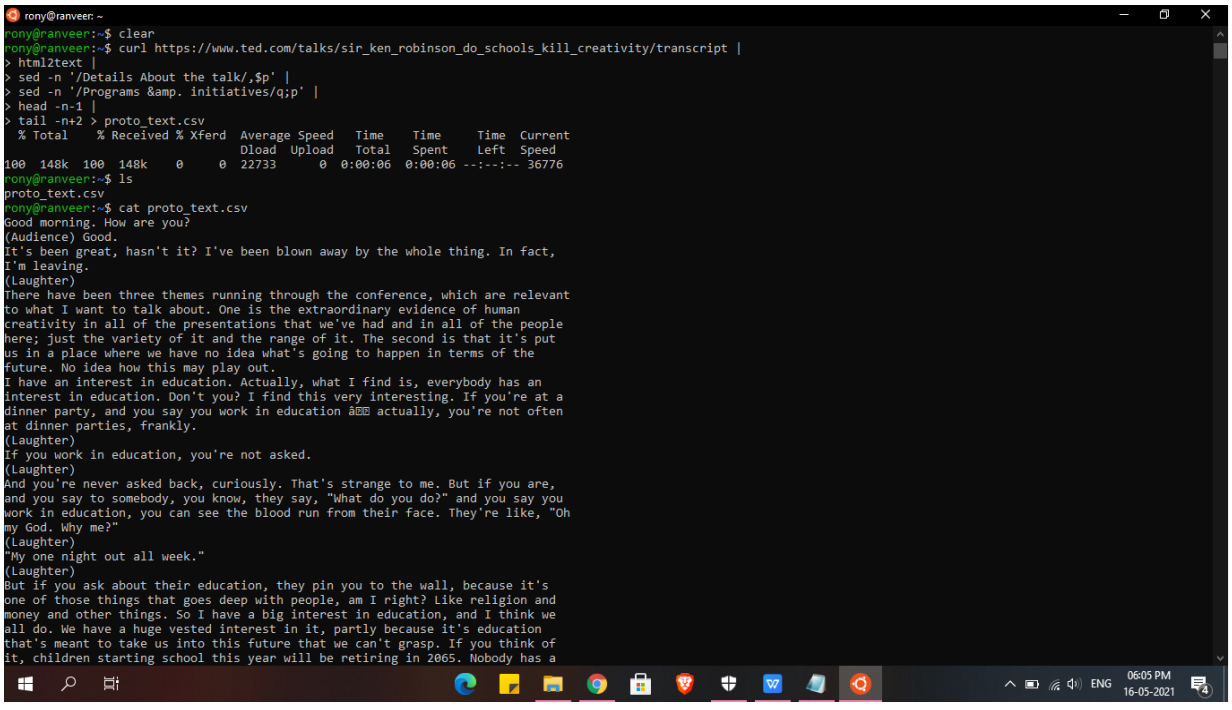
```
curl https://www.ted.com/talks/sir_ken_robinson_do_schools_kill_creativity/transcript | html2text |sed -n '/Details About the talk/,$p' | sed -n '/Programs &amp. initiatives/q;p'
```

5. To save this into a file first, so you will be able to reuse the data you got in the future.

```
curl https://www.ted.com/talks/sir_ken_robinson_do_schools_kill_creativity/transcript | html2text | sed -n '/Details About the talk/,$p' |sed -n '/Programs &amp. initiatives/q;p' | head -n-1 | tail -n+2 > proto_text.csv
```

6. To see

```
cat proto_text.csv
```



- **Extracting URLs from a listing page**

Scraping multiple pages(URLs) – using a for loop

1. The header of the for loop will be very similar to the one that you have learned at the beginning of this article:

   `for i in {1..107}`

   A slight tweak: now, we have 107 pages — so (obviously) we'll iterate through the numbers between `1` and `107`.

2. **Then add the `do` line.**

3. **The body of the loop** will be easy, as well. Just reuse the commands that you have already written for the first listing page a few minutes ago. But make sure that you apply the little trick with the page parameter in the URL! So it's not:

   ```
   https://www.ted.com/talks?language=en&page=1&sort=popular
   ```

   but:

   ```
   https://www.ted.com/talks?language=en&page=$i&sort=popular
   ```

   This will be the body of the for loop:

   ```
   curl "https://www.ted.com/talks?language=en&page=$i&sort=popular" |
   grep "href='/talks/" |
   sed "s/^.*href='/https\:\/\/www\.ted\.com/" |
   sed "s/?.*$/\/transcript/" |
   uniq
   ```

4. **Then add the `do` line.**

   All together, the code will look like this:

   ```
   for i in {1..107}
   do
    curl "https://www.ted.com/talks?language=en&page=$i&sort=popular" |
    grep "href='/talks/" |
    sed "s/^.*href='/https\:\/\/www\.ted\.com/" |
   ```

```
 sed "s/?.*$/\/transcript/" |

 uniq

Done
```

You can test this out in your Terminal… but in its final version let’s save the output of it into a file called ted_links.txt, too!

Here:

```
for i in {1..107}

do

 curl "https://www.ted.com/talks?language=en&page=$i&sort=popular" |

 grep "href='/talks/" |

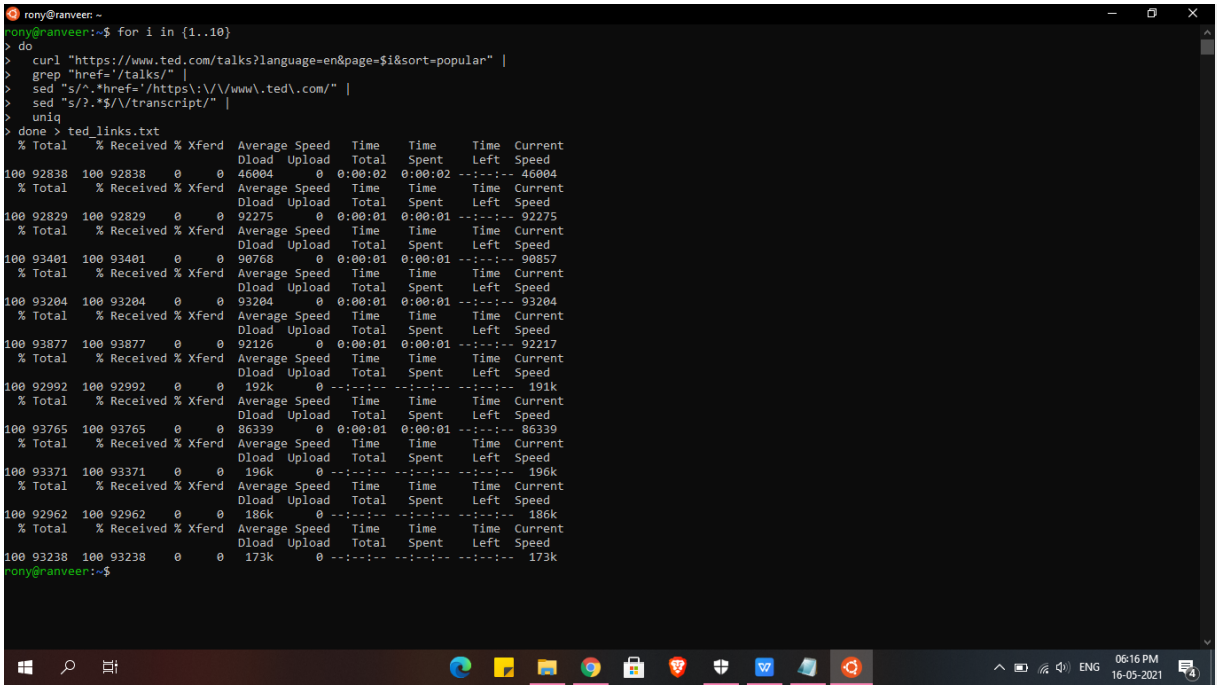
 sed "s/^.*href='/https\:\/\/www\.ted\.com/" |

 sed "s/?.*$/\/transcript/" |

 uniq

done > ted_links.txt
```

Now print the ted_links.txt



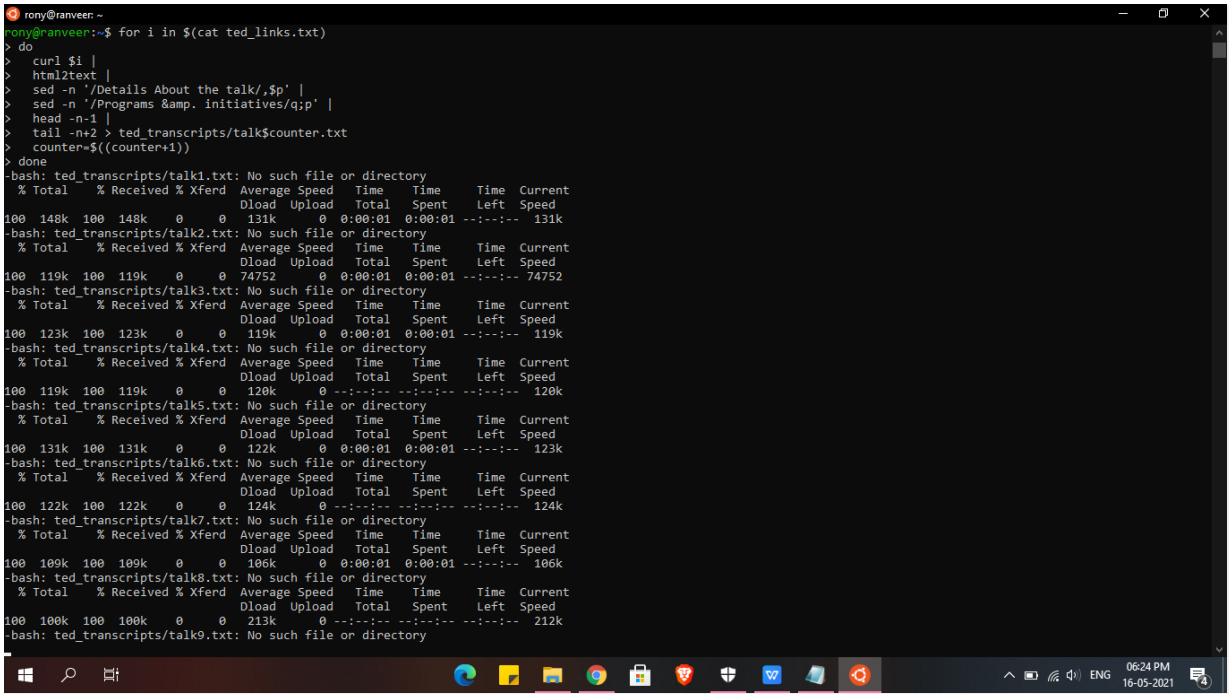
```
rony@ranveer:~$ ls
proto_text.csv  ted_links.txt
rony@ranveer:~$ cat ted_links.txt
https://www.ted.com/talks/sir_ken_robinson_do_schools_kill_creativity/transcript
https://www.ted.com/talks/james_veitch_this_is_what_happens_when_you_reply_to_spam_email/transcript
https://www.ted.com/talks/amy_cuddy_your_body_language_may_shape_who_you_are/transcript
https://www.ted.com/talks/simon_sinek_how_great_leaders_inspire_action/transcript
https://www.ted.com/talks/brene_brown_the_power_of_vulnerability/transcript
https://www.ted.com/talks/tim_urban_inside_the_mind_of_a_master_procrastinator/transcript
https://www.ted.com/talks/julian_treasure_how_to_speak_so_that_people_want_to_listen/transcript
https://www.ted.com/talks/bill_gates_the_next_outbreak_we_re_not_ready/transcript
https://www.ted.com/talks/sam_berns_my_philosophy_for_a_happy_life/transcript
https://www.ted.com/talks/robert_waldinger_what_makes_a_good_life_lessons_from_the_longest_study_on_happiness/transcript
https://www.ted.com/talks/cameron_russell_looks_aren_t_everything_believe_me_i_m_a_model/transcript
https://www.ted.com/talks/graham_shaw_why_people_believe_they_can_t_draw/transcript
https://www.ted.com/talks/mary_roach_10_things_you_didn_t_know_about_orgasm/transcript
https://www.ted.com/talks/tom_thum_the_orchestra_in_my_mouth/transcript
https://www.ted.com/talks/pamela_meyer_how_to_spot_a_liar/transcript
https://www.ted.com/talks/apollo_robbins_the_art_of_misdirection/transcript
https://www.ted.com/talks/susan_cain_the_power_of_introverts/transcript
https://www.ted.com/talks/david_blaine_how_i_held_my_breath_for_17_minutes/transcript
https://www.ted.com/talks/jill_bolte_taylor_my_stroke_of_insight/transcript
https://www.ted.com/talks/chimamanda_ngozi_adichie_the_danger_of_a_single_story/transcript
https://www.ted.com/talks/dan_pink_the_puzzle_of_motivation/transcript
https://www.ted.com/talks/jon_ronson_strange_answers_to_the_psychopath_test/transcript
https://www.ted.com/talks/kelly_mcgonigal_how_to_make_stress_your_friend/transcript
https://www.ted.com/talks/mel_robbins_how_to_stop_screwing_yourself_over/transcript
https://www.ted.com/talks/elon_musk_the_future_we_re_building_and_boring/transcript
https://www.ted.com/talks/james_veitch_the_agony_of_trying_to_unsubscribe/transcript
https://www.ted.com/talks/celeste_headlee_10_ways_to_have_a_better_conversation/transcript
https://www.ted.com/talks/shawn_achor_the_happy_secret_to_better_work/transcript
https://www.ted.com/talks/angela_lee_duckworth_grit_the_power_of_passion_and_perseverance/transcript
https://www.ted.com/talks/chris_anderson_ted_questions_no_one_knows_the_answers_to/transcript
https://www.ted.com/talks/alex_gendler_can_you_solve_the_prisoner_hat_riddle/transcript
https://www.ted.com/talks/jeff_dekofsky_the_infinite_hotel_paradox/transcript
https://www.ted.com/talks/hyeonseo_lee_my_escape_from_north_korea/transcript
https://www.ted.com/talks/elizabeth_gilbert_your_elusive_creative_genius/transcript
https://www.ted.com/talks/michael_mauser_what_are_those_floaty_things_in_your_eye/transcript
https://www.ted.com/talks/dan_gilbert_the_surprising_science_of_happiness/transcript
https://www.ted.com/talks/monica_lewinsky_the_price_of_shame/transcript
https://www.ted.com/talks/pranav_mistry_the_thrilling_potential_of_sixthsense_technology/transcript
https://www.ted.com/talks/keith_barry_brain_magic/transcript
https://www.ted.com/talks/adam_grant_the_surprising_habits_of_original_thinkers/transcript
https://www.ted.com/talks/maz_jobrani_a_saudi_an_indian_and_an_iranian_walk_into_a_qatari_bar/transcript
```

5.

```
mkdir ted_transcripts
```

6.

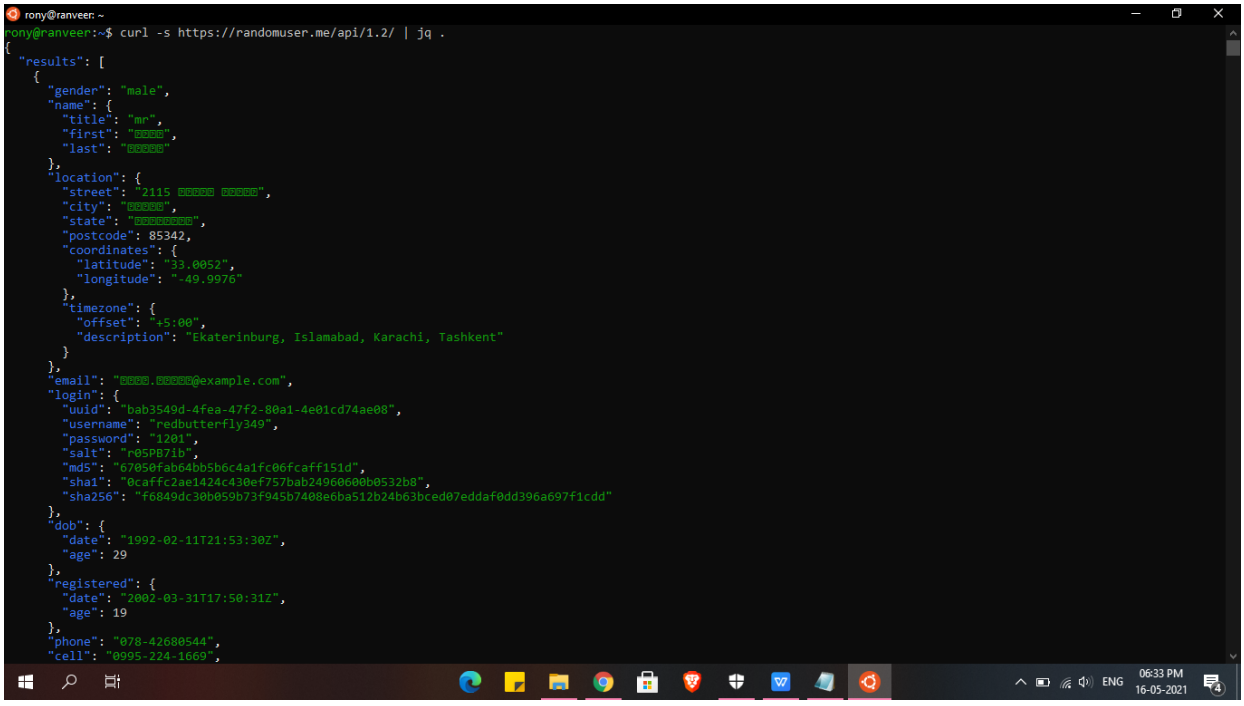
```
rony@ranveer:~$ for i in $(cat ted_links.txt)
> do
>   curl $i |
>   html2text |
>   sed -n '/Details About the talk/,$p' |
>   sed -n '/Programs &amp. initiatives/q;p' |
>   head -n-1 |
>   tail -n+2 > ted_transcripts/talk$counter.txt
>   counter=$((counter+1))
> done
-bash: ted_transcripts/talk1.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  148k  100  148k    0     0   131k      0  0:00:01  0:00:01 --:--:--  131k
-bash: ted_transcripts/talk2.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  119k  100  119k    0     0  74752      0  0:00:01  0:00:01 --:--:-- 74752
-bash: ted_transcripts/talk3.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  123k  100  123k    0     0   119k      0  0:00:01  0:00:01 --:--:--  119k
-bash: ted_transcripts/talk4.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  119k  100  119k    0     0   120k      0 --:--:-- --:--:-- --:--:--  120k
-bash: ted_transcripts/talk5.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  131k  100  131k    0     0   122k      0  0:00:01  0:00:01 --:--:--  123k
-bash: ted_transcripts/talk6.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  122k  100  122k    0     0   124k      0 --:--:-- --:--:-- --:--:--  124k
-bash: ted_transcripts/talk7.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  109k  100  109k    0     0   106k      0  0:00:01  0:00:01 --:--:--  106k
-bash: ted_transcripts/talk8.txt: No such file or directory
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  100k  100  100k    0     0   213k      0 --:--:-- --:--:-- --:--:--  212k
-bash: ted_transcripts/talk9.txt: No such file or directory
```



All you will get all the transcripts into one big file (ted_transcripts_all.txt).

6.2 Able to Make dozens of API calls and transform their output

Web APIs are not meant to be presented in nice layout, such as websites. Instead, most web APIs return data in a structured format, such as JSON or XML. Having data in a structured form has the advantage that the data can be easily processed by other tools, such as jq. For example, the API from https://randomuser.me returns data in the following JSON structure.



Another Example

```
$ curlicue-setup \

> 'https://api.twitter.com/oauth/request_token' \

> 'https://api.twitter.com/oauth/authorize?oauth_token=$oauth_token' \

>  credentials$ curlicue -f credentials \

> 'https://api.twitter.com/1/statuses/home_timeline.xml'
```

Important link:

https://advancedweb.hu/how-to-bash-and-jq-generate-statistics-for-a-rest-api/